Optimus: Warming Serverless ML Inference via Inter-Function Model Transformation

Paper Information

Title: Optimus: Warming Serverless ML Inference via Inter-Function Model Transformation

Authors: Zicong Hong (Hong Kong Polytechnic University), Jian Lin (Shantou University), Song Guo (The Hong Kong University of Science and Technology), Sifu Luo， Wuhui Chen(Sun Yat-sen University), Roger Wattenhofer (ETH Zurich), Yue Yu (Peng Cheng Laboratory)

Conference: EuroSys '24, April 22–25, 2024, Athens, Greece

Research Background

无服务器计算的兴起：

无服务器计算是一种新兴的云计算范式，它允许开发者构建由小型、无状态函数组成的应用程序，这些函数按需运行，无需专门的云服务器实例或基础设施管理。云提供商根据需求自动扩展资源，并仅按使用量收费，从而加快了产品上市时间并降低了成本。

机器学习推理服务的挑战：

随着机器学习（ML）的普及，将训练好的模型部署到生产环境中进行推理的需求日益增长。

在无服务器环境中进行 ML 推理面临的一个主要挑战是冷启动问题，即首次调用函数时需要加载模型和初始化环境，导致推理延迟较长。

模型加载延迟的瓶颈：

在无服务器 ML 推理中，模型加载是导致延迟的主要原因。

模型结构加载占据了模型加载延迟的大部分，而模型结构的加载可以重用已存在于容器中的其他模型的结构状态。

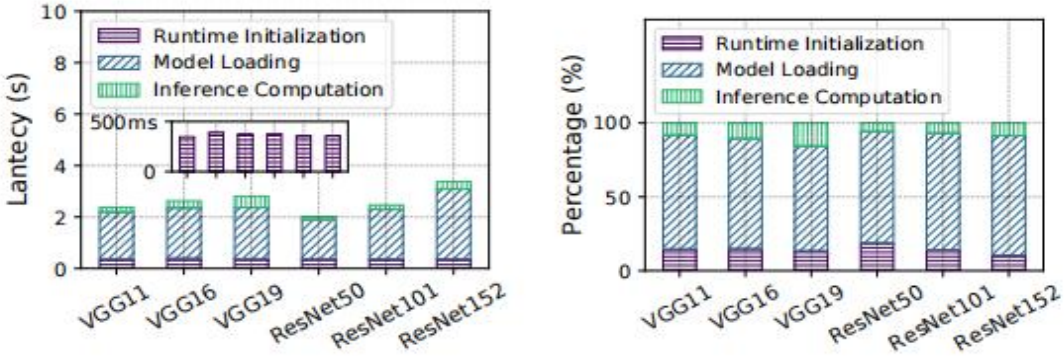
现有解决方案的不足：

现有的无服务器计算解决方案在处理 ML 推理时表现不佳，因为它们没有针对 ML 模型的特殊加载需求进行优化。

一些工作尝试通过跨函数容器共享来减少冷启动延迟，但它们没有深入到模型操作的粒度来进行模型转换，因此无法充分利用容器内已有模型的结构状态。

Motivation

见解 1：模型加载延迟在无服务器 ML 推理函数的总延迟中占主导地位。



(a) Request processing time

(b) Every step proportion

Model	VGG11	VGG16	VGG19	ResNet50	ResNet101	ResNet152
Params	132.9M	138.4M	143.7M	25.6M	44.7M	60.4M
Size (MB)	506	528	549	98	171	232

(c) Number of parameters and size of varying models

Figure 2. Request processing time for varying models in serverless ML inference.

见解 2：模型结构加载延迟在无服务器 ML 推理函数的模型加载延迟中占主导地位。

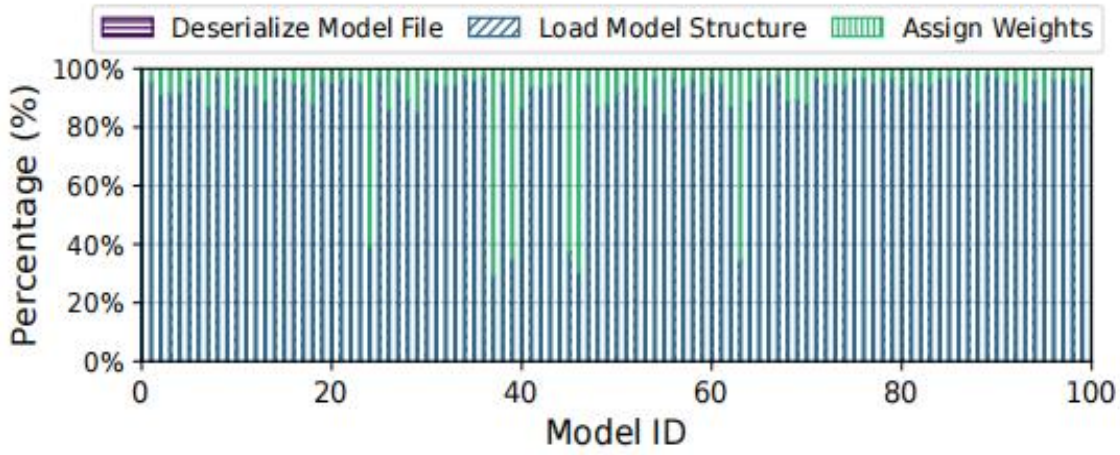


Figure 3. Latency of each step in the model loading for 100 models from Imgclsmob [37] in serverless ML inference.

见解 3：结构相似的模型在现有的机器学习应用中很常见。

在许多模型库中，结构相似的模型也很普遍，例如 NASBench、Imgclsmob 和 HuggingFace。这是因为大多数模型都依赖于相似的结构设计，但具有更宽/更深的层、分支或从不同数据集训练得到的不同权重。例如，卷积（CONV）和注意力操作分别在计算机视觉（CV）和自然语言处理（NLP）模型中得到了广泛应用。

System Overview

1. Idle Container Identification Mechanism
2. In-container Transformation Meta-operators
3. Scheduling Algorithm for Inter-function Model Transformation

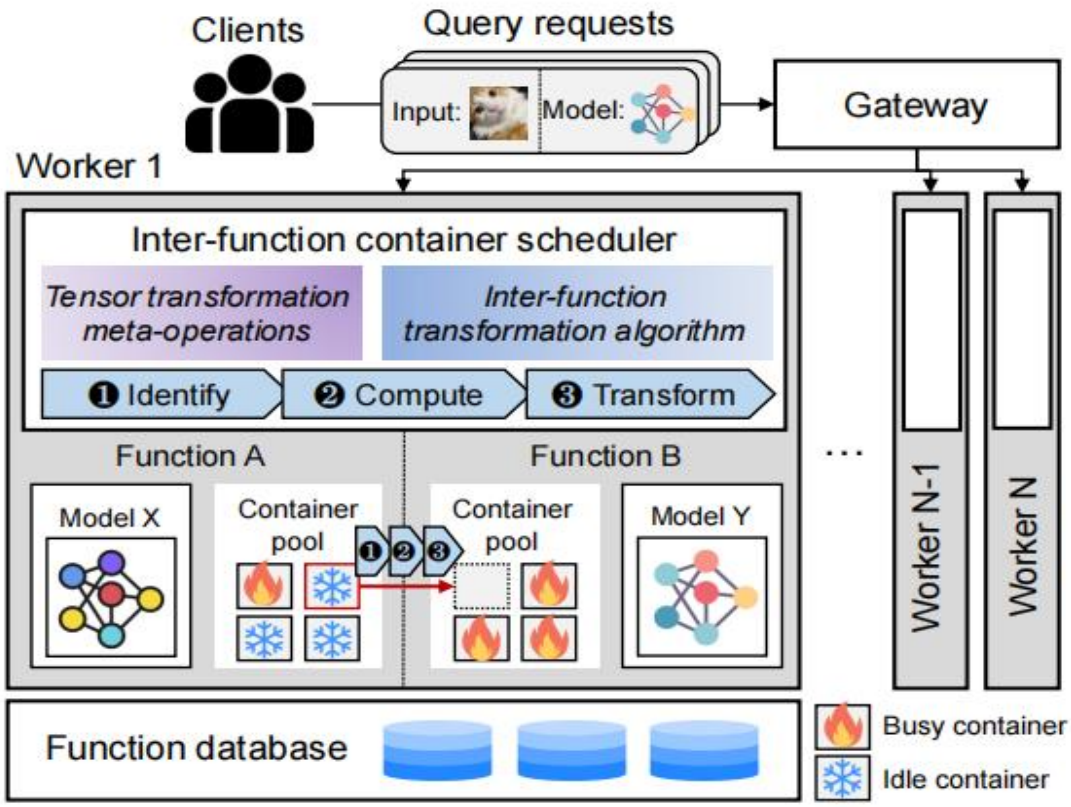


Figure 6. System architecture overview of OPTIMUS.

Idle Container Identification Mechanism

为了识别每个节点中的空闲容器，Optimus 为每个容器采用了一种计时器设计。当一个新请求被路由到一个容器时，该容器的计时器会被重置为 0。当计时器超过一个预定义的阈值（例如 60 秒）时，该容器被视为空闲，并且其中维护的模型可以转换为其他缺少预热容器的功能的模型。

In-container Transformation Meta-operators

替换：对于仅替换权重而保留操作结构的转换。

重塑：如果源模型和目标模型中的两个操作类型相同但属性不同（例如，对于卷积操作，其核大小、核数量和步长不同），我们提出了一个元操作符“重塑”，它可以在不重新生成新操作的情况下修改这些属性。

减少：对于源模型中无法与目标模型中任何操作匹配的操作，我们提出了一个元操作符“减少”，用于删除这些操作而不影响其他操作。

添加：如果源模型中的任何操作无法通过元操作符“替换”和“重塑”转换为目

标模型，我们提出了一个元操作符“添加”，用于在源模型中添加一个新操作。

边：为了改变/删除/添加任意两个操作之间的边，我们提出了一个元操作符“边”。

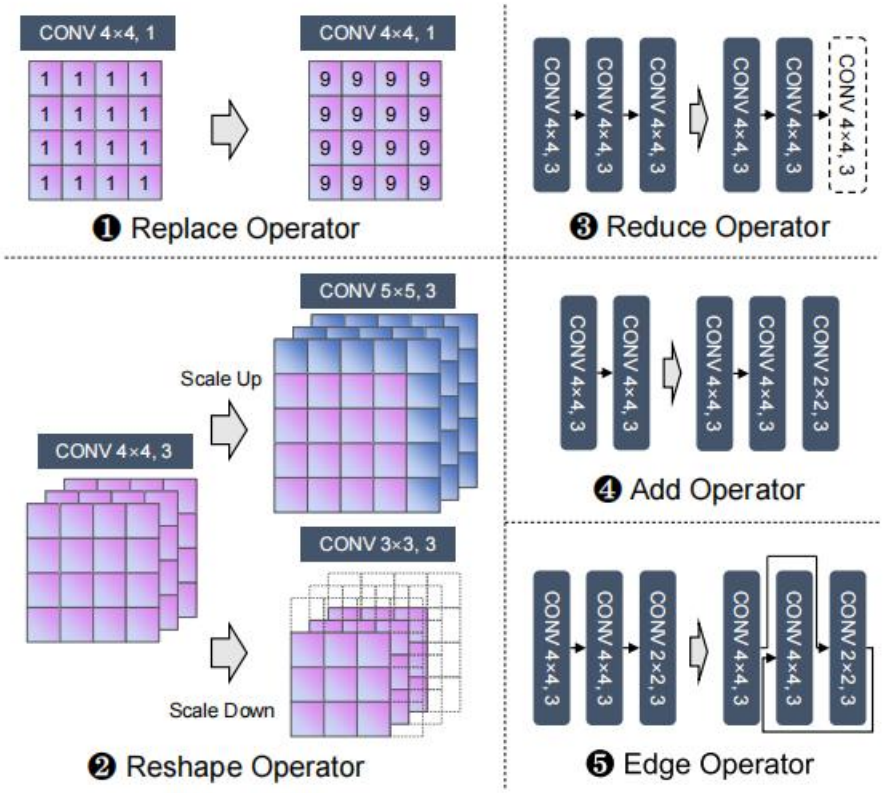


Figure 7. Illustration of in-container transformation meta-operators. For a CONV operation represented by $x \times y, k$, $x \times y$ denote kernel size, and k denotes the number of kernels.

Scheduling Algorithm for Inter-function Model Transformation

Module 1: Offline Profiling for Meta-operators

1. 元操作符“替换”的执行时间取决于目标模型操作中权重的大小。

2. 元操作符“添加”的执行时间取决于目标模型操作的类型和属性。

3. 元操作符“重塑”的执行时间取决于目标操作形状变化的幅度。

4. 元操作符“减少”的执行时间是恒定的，而元操作符“边”的执行时间则可以忽略不计。

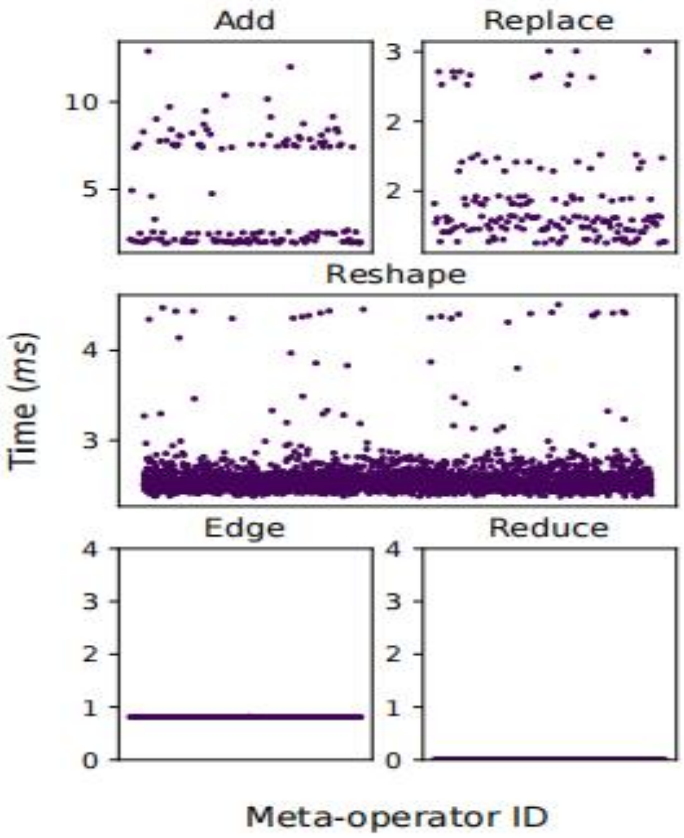


Figure 8. Execution time of varying meta-operators.

Module 2: Planning for Model Transformation

给定两个模型 g1 和 g2，我们定义了一系列能够将 g1 转换为 g2 的元操作符序列，表示为 e = {e1, e2, ...}。其中，ei 是 e 中的一个元素。其预估成本 c(ei)可以通过离线分析收集得到。因此，序列 e 的成本定义为 Σc(ei)（ei 属于 e）。我们将所有从 g1 到 g2 的可行元操作符序列的集合表示为 E(g1, g2)。注意，序列 e 中元操作符的顺序不会改变成本。我们的目标是找到一个最优的元操作符序列 e，满足 e*属于 argmin Σc(ei)（ei 属于 e，e 属于 E(g1, g2)），即成本最小的序列。

Module 2+: Efficient Planning Algorithm

(1) 根据第一个观察结果，我们首先根据类型将源模型（或目标模型）中的所有操作进行分组。

(2) 我们在源模型和目标模型中相同操作类型的两组操作之间，按顺序逐一匹配两个操作。这种启发式设计是基于上面提到的第二个和第三个观察结果。

(3) 两个匹配操作之间的转换是通过元操作符“替换”或“重塑”来实现的。

(4) 之后，如果源模型中的一组操作中有未匹配的操作，我们将通过元操作符“减少”来删除它们。

(5) 如果目标模型中的一组操作中有未匹配的操作，我们将通过元操作符“添加”来创建它们。

(6) 最后，我们可以使用元操作符“边”来修改模型中的数据流。

Module 3: Online model transformation execution

规划策略缓存：当一个新的模型在 Optimus 的全局模型仓库中注册时，系统会测量新模型与仓库中现有模型之间的转换开销，并基于规划算法缓存转换策略。当模型转换请求到达时，管理器可以读取缓存的转换策略，并据此进行模型转换。

保障设计：在某些情况下，我们跨功能的模型转换开销高于从头加载模型的开销。因此，在这些情况下，系统会像传统方式那样从头加载一个新模型。换句话说，在最坏的情况下，Optimus 的性能也能得到保证。

Design Refinement

Model Sharing-aware Load Balancer

1.如果一个节点上函数的模型结构差异很大，那么该节点内的跨函数模型转换将会非常昂贵。

2.如果一个节点上函数的需求动态相似，那么很少有空闲容器能够帮助那些遭遇冷启动问题的函数。

3.在 Optimus 中，需要一个新的无服务器负载均衡器，该负载均衡器需要考虑模型结构的相似性和需求动态的互补性。

Solutions

1.所提出的调度器旨在通过 K-medoids 聚类算法，在同一节点上部署具有相似模型结构但需求动态不同的函数。

2.它将函数视为不同的点，并根据它们的模型编辑距离和需求动态差异来测量任意两个函数之间的距离。

3.我们基于它们历史记录的协方差来计算它们需求动态的互补性 K(A, B)。根据它们在 T 个时间段内的历史需求动态{ltA}t∈T 和 {ltB}t∈T，我们可以得到 K(A, B)。

Experimental Setup

节点设置：我们使用两台服务器来评估 Optimus：一台配备了具有 104 个核心和 128GB 内存的 Intel Xeon Gold 5320 CPU，另一台配备了具有 48 个核心、64GB 内存和 4 个 NVIDIA GeForce GTX 1080 Ti 显卡的 Intel Xeon E5-2650 v4 CPU。这些机器通过 10 Gbps 全双工带宽以太网相互连接。服务器运行的是 Ubuntu 22.04.2 和 Docker 23.0.1。

工作负载：Imgclsmob、BERT、NASBench。

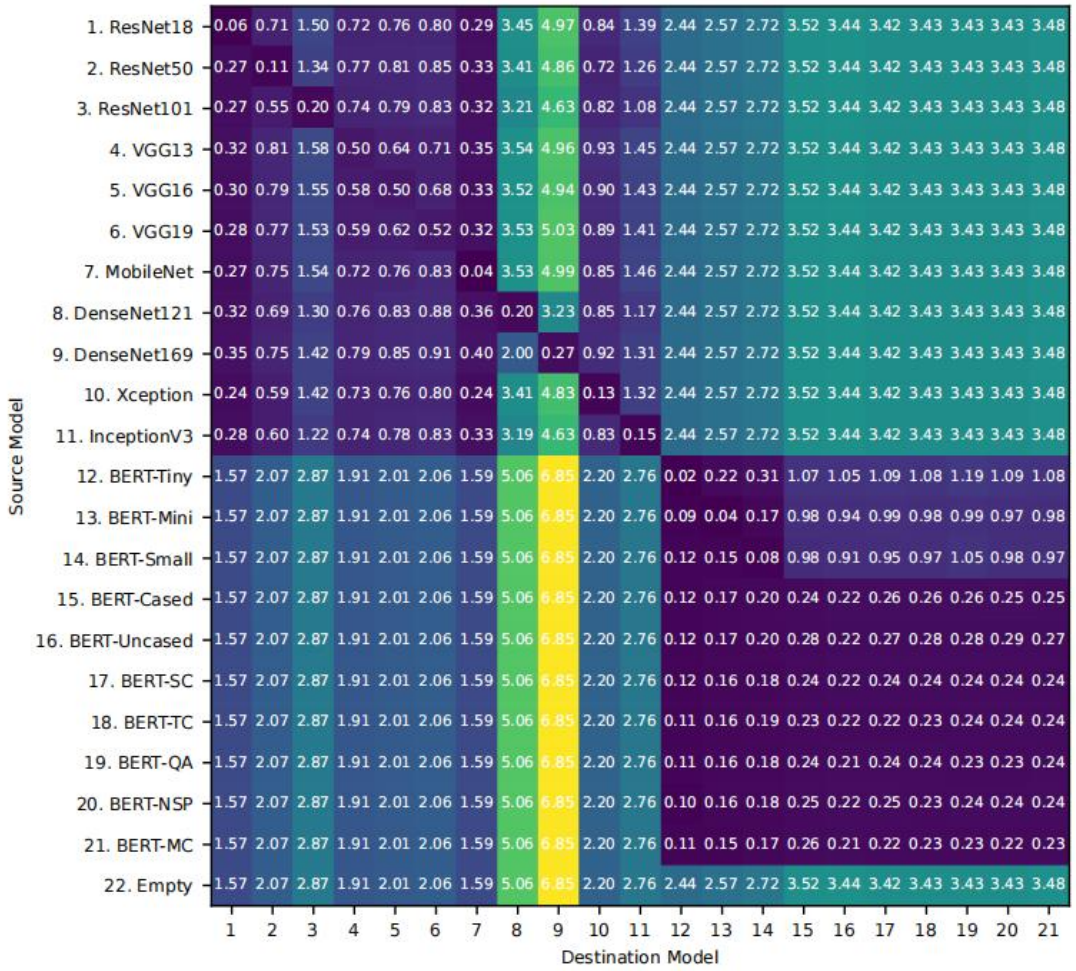
工作负载的函数调用到达模式

泊松分布：我们按照泊松分布向每个无服务器推理服务发送查询。为了模拟频繁、中等和不频繁的工作负载，我们将 λ 分别设置为 10^-3.5、10^-2、10^-2.5。

Azure Function：为了模拟类似生产环境中的工作负载到达模式和特征，我们使用了 2021 年从 Microsoft 生产系统中收集的两周 Microsoft Azure Function 跟踪数据。

对比系统：OpenWhisk、Pagurus、Tetris。

Inter-function Model Transformation



1. 与现有无服务器机器学习推理系统从头加载模型相比，跨函数模型转换可将延迟降低高达 99.08%。在运算类型和规模方面，结构更相似的模型之间的转换所需时间更少。

2. 在大多数情况下，从大模型转换到小模型的速度比从小模型转换到大模型的速度要快。

3. 结构相同但权重不同的模型之间的转换所需时间最少。

Serverless ML Inference Latency

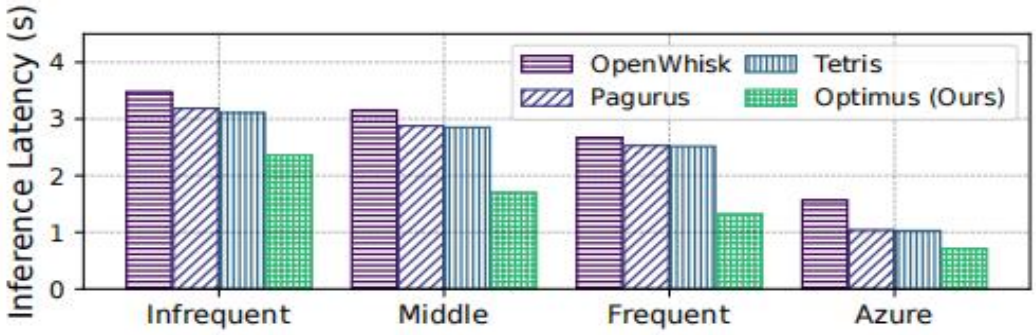


Figure 13. Average service time of serverless ML inference requests under the Poisson and Azure workloads.

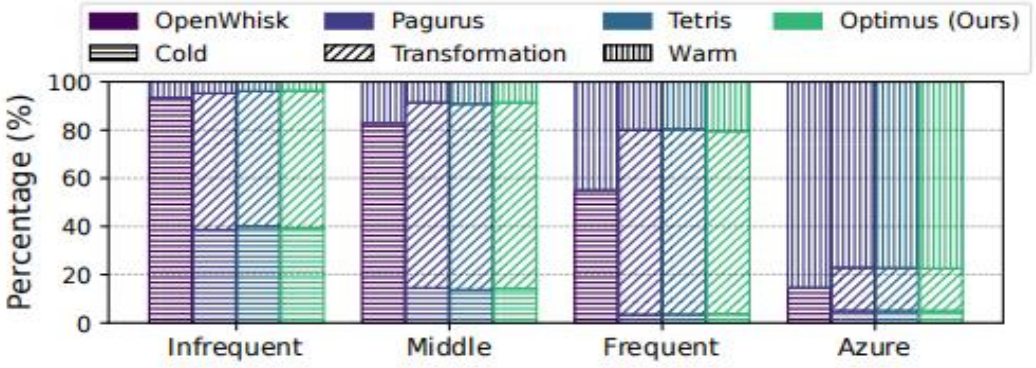


Figure 14. Percentage of cold start, model transformation, and warm start of serverless ML inference requests under the Poisson and Azure workloads.

Conclusion

1.我们推出了 Optimus，这是一个基于全新跨函数模型转换理念的低冷启动开销的新型无服务器机器学习推理系统。

2.我们实现了一个原型，该原型支持在卷积神经网络（CNN）和转换器（transformer）模型上进行无服务器机器学习推理查询。

3.结果显示，与现有的无服务器机器学习推理系统相比，Optimus 在泊松模拟工作负载和来自 Azure 的真实工作负载上，平均服务时间减少了 24.00%至 47.56%。